# Great Algorithms: RSA

Aaron Gorenstein

September 1, 2013

**Abstract**

The RSA cryptosystem is really extraordinary, and I cannot fault anyone for teaching it to undergraduates. However, all of the resources I found taught RSA from the "bottom up"—first number theory, then RSA. It was hard for me and others to piece that together, so I wrote this to present RSA in a "top-down" fashion, with only the smallest amount of necessary modular arithmetic. The presentation is my own, but all of the proofs and reasoning are from [DPV08, CLRS01], and I will cite to clarify which in the body.

I assume you've given some effort understanding the RSA algorithm from one of those sources. This is because I'm a bit casual with my definitions and motivation in the text.

## 1 A Tiny Bit of Modular Arithmetic

Integer arithmetic (essentially addition and multiplication) is defined over an infinite set. But what if you only have a finite set of integers? This naturally comes up in computer science (if integers are only 64 bits or what-have-you). We still want addition and multiplication, but ones that "work" even if we run into the MAX_INT value.

We define *modular arithmetic* in this simple way: say $N$ is the largest integer. Then, do whatever arithmetic operation(s) you'd like to get the result $r$. Then, subtract from $r$ as many copies of $N$ as needed until $0 \leq r < N$. You can do this "shrinking" at any point, and as many times as you'd like, during your computation.

### 1.1 A simple example

Here's a concrete example, also to introduce our notation. For instance, if you have $(x+y) \cdot z \pmod{N}$, that means we want to talk about the result of that arithmetic, but with the fact that $N$ is the largest integer we'll allow. So, say that $x = 14, y = 2, z = 6, N = 11$. Then, we see that $x = 14 > N$, so we replace it with $x - N$ and get $(3 + 2) \cdot 6 \pmod{N}$. Then let's multiply by 6: we get $18 + 12 \pmod{N}$. But both of those are too big again! So we replace those values again: $(18 - 11) + (12 - 11)$, and get 8.

The key fact is that we could these "shrinking operations" whenever, and *still* get 8. Let's say we save it until the end: then we get $16 * 6 = 96$. Clearly we can subtract $N = 11$ from this 8 times, and we get $96 - 88 = 8$. Amazing, no?

### 1.2 Key Concept: Equivalence "mod" $N$

In the previous example, we would say that we were computing things "mod $N$" or "mod eleven". This fascinating property that we can "shrink" by $N$ at whatever stage in the arithmetic we'd like is expressed by the idea of *equivalence* in modular arithmetic. Formally, if we write:

$$x \equiv y \pmod{N} \tag{1}$$

to say that $x$ is equivalent to $y$ mod (shrinking-by) $N$. What this means in real-life arithmetic is that there is some integer $k$ such that

$$x = y + kN. \tag{2}$$

Note the actual equality sign used there. Going back to our example, we can say that $14 \equiv 3 \pmod{11}$. What that means is that, when $N = 11$, we can interchange any instance of the value 3 with 14 (or vice-versa). This is the case because $14 = 3 + 11 \cdot 1$. In a similar vein, we know that $96 \equiv 8 \pmod{N}$ because $96 - 8 \cdot N = 8$ (in this case, $N = 11$ and $k = 8$).

That we can compute modular arithmetic without regards to when we "shrink" the values according to $N$ is based on the following two equivalences:

$$x + y = z \implies x + y \equiv z \pmod{N} \tag{3}$$

$$x \cdot y = z \implies x \cdot y \equiv z \pmod{N}. \tag{4}$$

These statements are important when, implicitly, $z > N$. To prove these statements is a common exercise, but for brevity those proofs are not included. (Note that because multiplication "works" mod $N$ as expressed by eq. (4), so too does exponentiation, as it is just defined in terms of multiplication.) (Further note that we can extend this whole notion to negative numbers: to get $-x \pmod{N}$, simply add $N$ to $-x$ enough times until the resulting value $0 \leq kN - x < N$.)

An important idea is **modular inverse**. You may have noticed that $6 \cdot 2 \equiv 1 \pmod{11}$. The number 1 is important because, well, it's 1! Obviously, $x \cdot 1 \equiv x \pmod{N}$, which can come in handy. More pragmatically, if we know $x \cdot y \equiv 1 \pmod{N}$, and we have $a \cdot x \equiv b \pmod{N}$, we know that $a \equiv by \pmod{N}$. Explicitly:

| | | |
|---|---|---|
| $x \cdot y \equiv 1 \pmod{N}$ | By assumption | (5) |
| $a \cdot x \equiv b \pmod{N}$ | By assumption | (6) |
| $a \cdot x \cdot y \equiv b \cdot y \pmod{N}$ | We just mult. both sides by $y$ | (7) |
| $a \cdot 1 \equiv b \cdot y \pmod{N}$ | As $x \cdot y \equiv 1 \pmod{N}$, we do that. | (8) |

We call $x$ the "inverse of $y \pmod{N}$" in that case. (And of course, we can also say $y$ is the inverse of $x \pmod{N}$.) Note that inverses are dependent on the value of $N$!

Lastly, remember these two final equivalences: $N \equiv 0 \pmod{N}$ (and by extension, $kN \equiv 0 \pmod{N}$). Furthermore, if $x = y$, then surely $x \equiv y \pmod{N}$.

### 1.3 Equivalence for Encryption

Now that we understand, at least on a functional level, what equivalence is, why do we care about this for the RSA algorithm? The fundamental concept is this: say we have some message $x < N$. (That is to say, we have some message that is a binary string, which we interpret as a number. Then we choose some really big $N$.) Then we encrypt $x$ by the function $f$, and we get $y = f(x)$, but $y > N$. Our encryption scheme will transmit $y' < N, y' \equiv y \pmod{N}$. Thus, $y'$ is the encrypted version of the message $x$. This is troubling, because intuitively we have "shrunk" $y$ into $y'$, perhaps losing information.

However, as we've established, arithmetic operations "work" in the modular universe. Say the decryption method $f^{-1}(y)$ is defined *only* in terms of addition and multiplication. Then, *even though we only have* $y'$, we can apply $f^{-1}(y')$ and get $f^{-1}(y) = x$. This is surprising!

## 2 The Algorithm

The algorithm is worryingly simple. Alice chooses two large primes: $p, q$ and defines $N = p \cdot q$. Alice also chooses a power $e$. Lastly, Alice computes the *inverse* of $e \pmod{(p-1)(q-1)}$ called $d$. In other words, she computes some value $d$ such that $ed \equiv 1 \pmod{(p-1)(q-1)}$. Then Alice publishes $(e, N)$ as her public key.

Bob gets $(e, N)$ and wants to send some message $x < N$ to Alice. He sends $y' \equiv x^e \pmod{N}$ to Alice. That's all Bob does!

Lastly, Alice decodes Bob's message $y'$ by this simple arithmetic: we claim that $y'^d \equiv x \pmod{N}$. That's it! She simply powers $y'$ to the $d$-th power and then takes it modulo $N$.

Concretely, the encryption is multiplying $x$ to itself $e$ times. Then we multiply the resulting $y'$ by itself $d$ times, and we claim we get $x$ back! (Obviously, this is only possible when we work in a modular universe.) In modular-arithmetic terms, what we're really claiming is this:

$$x^{ed} \equiv x \pmod{N}. \tag{9}$$

That's it! That's the whole thing. Recall that $x^e$ is the encryption of $x$, and raising that encrypted value to the $d$-th power $\pmod{N}$ is the way to decrypt it. So we compress those ideas in eq. (9). The only thing left is proving that equation is correct!

# 3 The Correctness

## 3.1 Two Quick Theorems

We will use two theorems, which I will not prove:

**Theorem 1** (Fermat's Little Theorem)**.** *For any prime $p$ and any integer $a$:*

$$a^{p-1} \equiv 1 \pmod{p}. \tag{10}$$

**Theorem 2** (Chinese Remainder Theorem)**.** *For two values $p, q$ coprime with one another, and some integer $a$, if:*

$$a \equiv 1 \pmod{p} \tag{11}$$
$$a \equiv 1 \pmod{q} \tag{12}$$

*then*

$$a \equiv 1 \pmod{p \cdot q}. \tag{13}$$

(This is usually presented as a corollary to a much-more-general formulation of the Chinese Remainder Theorem, but this is all we need for the proof.)

The proofs of these two statements make key (and brilliant) use of the properties of modular arithmetic, and for that reason we'll skip them: we want to get straight to the RSA proof of correctness. Plus, they're often assigned as homework. In any case, here are the equations which will show eq. (9) correct:

$$x^e \equiv x' \pmod{N} \qquad \text{By assumption/definition.} \tag{14}$$
$$x^{ed} \equiv (x')^d \pmod{N} \qquad \text{By modular arithmetic.} \tag{15}$$
$$x^{ed} = x^{1+k(p-1)(q-1)} \qquad \text{By definitions of inverses } \pmod{k}. \tag{16}$$
$$x^{1+k(p-1)(q-1)} \equiv (x')^d \pmod{N} \qquad \text{Plugging in eq. (16) for eq. (15).} \tag{17}$$

The key trick is eq. (16): using the definition of modular equivalence, we know that if $ed \equiv 1 \pmod{(p-1)(q-1)}$, then there must be some integer $k$ such that $ed = 1 + k(p-1)(q-1)$. So, we simply replace $ed$ with that value. These ideas (the equality and the replacement) were covered in section 1.2.

With that final equation, eq. (17), established, we bring to bear those two theorems we stated:

$$(x^{k(p-1)})^{q-1} \equiv 1 \pmod{q} \qquad \text{By Fermat's theorem} \tag{18}$$
$$(x^{k(q-1)})^{p-1} \equiv 1 \pmod{p} \qquad \text{By Fermat's theorem} \tag{19}$$
$$x^{k(p-1)(q-1)} \equiv 1 \pmod{pq = N} \qquad \text{By the Chinese Remainder Theorem.} \tag{20}$$

Why do we care about about that $x^{k(p-1)(q-1)}$ value? Well, observe that $x^{1+k(p-1)(q-1)} = x \cdot x^{k(p-1)(q-1)}$. Let's use that in our final reasoning:

$$\begin{aligned} x^{ed} &\equiv x \cdot x^{k(p-1)(q-1)} \mod N && \text{By eq. (17)} && (21)\\ &\equiv x \cdot 1 \mod N && \text{By eq. (20)} && (22)\\ &\equiv x \mod N && \text{Our goal.} && (23) \end{aligned}$$

And that concludes the proof of correctness!

# 4 The Things Left Unsaid

The goal of this exposition was to trick you into thinking you understand how the algorithm works. More positively, I hope this illustrates the *outline* of the proof that RSA really works. The main conceptual gap is just a more rigorous understanding of modular arithmetic: I've glossed over the fact that multiplicative inverses don't always exist $\pmod N$ (but they do when $N$ is prime!) and the proofs that modular arithmetic really works. The key idea for modular arithmetic working stems from the fact that $x \equiv y \pmod N \implies x = kN + y$ for some $k$, and then $kN \equiv 0 \pmod N$. The two theorems take some proving. Lastly, there's the super-critical proof that breaking this encryption is actually hard! That's usually a homework assignment though. Regardless, a standard algorithms textbook has an excellent coverage of the RSA algorithm [DPV08], though the proof of correctness here more closely follows that of CLRS [CLRS01]. Look to them for more information!

# References


[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 2001.

[DPV08] S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani. *Algorithms*. McGraw-Hill Higher Education, 2008.